Menu Structures, Toolbars and the MDI

Creating a Menu Structure

A menu structure consists of three types of dialog elements:

- menu-bar controls,
- menu items,
- submenu controls.

A menu structure has one menu-bar control consisting of several menu items. The menu bar with its items is displayed directly beneath the window's title bar. Each menu item may be simple or may represent a submenu control, which allows you to pull down several menu items grouped vertically. Therefore, submenu controls may contain items representing a submenu control one level lower. A submenu control becomes visible when the representing item in the menu-bar control or the parent submenu control is clicked upon.

There are two ways to create menu structures:

- Use the Dialog Editor; or
- use Natural code.

If you use the Dialog Editor

1. Check the "Menu Bar" entry in the dialog's attribute window. Click OK.
   When you go back to the dialog, a dummy menu-bar control appears.
2. Double-click on the dummy menu-bar control, or from the Natural Menu, select "Dialog > Menu Bar", or use CTRL+M.
   The "Dialog Menu Bar" dialog box appears. This dialog box is divided into three group frames: menu bar, selected submenu and selected menu item.
3. In the selected menu items group frame, use "New" to add a menu item behind the selected position, or at the beginning. Now use the selected menu-item group frame to modify attribute values or add event handlers to the new menu item.

Normal menu items have a click event whose code is executed when the end user clicks on the menu item.

Note: The MENU-ITEM-TYPE of the menu item can also be "Separator", in which case the item is no text item.

If you use Natural code

1. Create a Menu Bar with the PARENT attribute set to 'NULL-HANDLE' or '*windowhandle*'.
2. To create a simple menu item: the PARENT attribute must have the value '*menubarhandlename*'.
3. To create a submenu control: the submenu control's PARENT attribute must have the value 'NULL-HANDLE' or '*windowhandlename*'. Then create a menu item with PARENT = '*menubarhandlename*' and MENU-HANDLE = *'submenuhandlename'*.
4. Then associate the menu bar with a dialog window by updating the window's MENU-HANDLE attribute with the handle of the menu bar as set in the first step.
5. The event handling for the dynamically created menu items must be done in the default event handler, as described in the section How to Create and Delete Dialog Elements Dynamically.

The PARENT attribute determines when the menu bar or the submenu control will be destroyed. When PARENT = '*windowhandlename*', the menu bar/the submenu control will be destroyed when the window is destroyed. This is the default setting, which is also used by the Dialog Editor. If PARENT = NULL-HANDLE, the menu bar/the submenu control will be destroyed only when the application is terminated.

If you define the menu structure's handles inside a global data area, you can share these definitions among several dialogs.

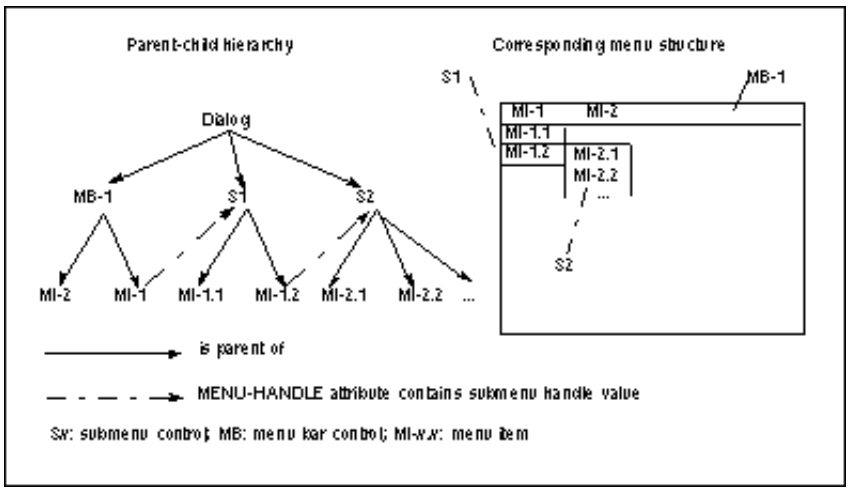
To build the above menu structure

1. Define the handles of the menu-bar control, the menu items, and the submenu control(s) as the user-defined variables in the handler of the applicable event.
2. Create the controls and items by assigning values to the attributes (PARENT, ...) and by executing the PROCESS GUI statement action ADD.
3. Create the controls and items in the sequence menu-bar control, submenu control with menu items.
4. Insert the controls and items in the sequence submenu control into menu-bar control, and menu-bar control into dialog window.

You can study how to build a menu structure in code by using the enhanced dialog list mode to list a dialog with an editor-built menu. To get a code model for creating a menu item, create a menu bar control with the Dialog Editor, go to the menu-bar control attributes window, cut a menu item and paste it into any chosen event-handler section. The generated code for the menu item appears.

Parent-Child Hierarchy in Menu Structures

Sometimes, it is necessary to use code for going through each element in a menu structure. For menus, the parent-child hierarchy is structured in a way that is not evident from the graphical representation of the menu structure.



In the above diagram, the first child of the dialog would be the menu-bar control. Its successor would be submenu control S1, and so on. To go from menu item MI-1 to submenu S1, you query for the MENU-HANDLE attribute value of MI-1. The value you get is the handle value of S1.

Creating a Toolbar

There are two ways of creating toolbars and their items:

- Use the Dialog Editor; or
- use Natural code to create them dynamically.

To use the Dialog Editor

1. Double-click on the toolbar or from the Natural Menu, select "Dialog > Toolbar".
   The toolbar attributes window opens.
2. Add toolbar items by clicking on the "New" push button.
3. Assign bitmap file names and other attribute values to the new toolbar item.

If you want to use Natural code for dynamic creation, you can study how to build a tool bar in code. Use the enhanced dialog list mode to list a dialog with an editor-built tool bar.

Sharing Menu Structures, Toolbars and DILs (MDI Application)

An MDI (multiple document interface) application consists of a frame dialog that provides the menu structure, toolbar, and DIL shared among all child dialogs. An MDI frame dialog allows you to tile or cascade its child dialogs.

Note: You may only share the toolbar if the PARENT of the toolbar is the dialog of the highest level (the main dialog of an application).

To create an MDI frame dialog

1. Use the Dialog Editor, and go to the dialog object’s attributes window.
2. Choose "MDI frame window" in the "Type" entry.

An MDI frame dialog must not contain dialog elements other than menu-bar control, submenu control, menu item, toolbar, and toolbar item.

To create an MDI child dialog

1. Use the Dialog Editor, and go to the dialog object’s attributes window.
2. Choose "MDI child window" in the "Type" entry.

An MDI child dialog:

- can be moved and sized only inside the area of their MDI frame dialog;
- can be maximized to the full size of the area of their MDI frame dialog;
- can be minimized, after which its icon appears at the bottom of its MDI frame dialog;
- can have its own menu structure, toolbar, and DIL. Those do not appear inside the child dialog but are displayed in the MDI frame dialog when the child dialog is active. When another MDI child dialog becomes active, the menu structure, toolbar, and DIL change at the same time;
- can be arranged in a tile or cascade by setting a menu item’s attribute MENU-ITEM-TYPE to the values "MDI Cascade" or "MDI Tile";
- can have its title added to the end of an MDI-WINDOWMENU type submenu control. By choosing one of these menu items, the corresponding MDI child dialog becomes active.

If you want to open an MDI child dialog from within an MDI frame dialog, you can, for example, create a menu item in a menu structure of an MDI frame dialog and define a click event for the menu item. You then write the OPEN DIALOG code for opening an MDI child dialog in the click event handler. The end user will open the MDI child dialog from within the MDI frame dialog by clicking on the menu item, triggering the click event handler.

Example:

```
OPEN DIALOG ’MDICHILD’ #DLG$WINDOW #CHILD-ID
```

The first operand is the name of the dialog created by the Dialog Editor by selecting "MDI child window" in the "Type" selection box. The second operand is the parent of the new MDI child dialog. This must be the MDI frame dialog. The third operand is a Natural variable defined as I4 in the dialog’s data areas. This variable receives the

dialog ID returned by the system.

Note: #DLG$WINDOW is a generated variable.

You can also open an MDI child dialog from within another MDI child dialog (open a sibling of your MDI child dialog). Then you write a similar click-event handler as above:

Example:

```
OPEN DIALOG 'MDICHILD' #DLG$PARENT #CHILD-ID
```

The first and the third operands are the same as above. The second operand must be the parent of both MDI child dialogs.

Note: #DLG$PARENT is a generated variable.

Back to Event-Driven Programming Techniques.

